CONTROL SYSTEM USING PLURAL OBJECTS, CONSTRUCTING METHOD THEREFOR AND PERIPHERAL EQUIPMENT CONTROL SYSTEM

Patent Number: JP9106355
Publication date: 1997-04-22
Inventor(s): KIMURA YOSHIHIRO; HISAMATSU YUTAKA
Applicant(s):: SEIKO EPSON CORP
Requested Patent: ⌐ JP9106355
Application Number: JP19950264193 19951012
Priority Number(s):
IPC Classification: G06F9/46 ; G06F9/06 ; G06F9/44 ; G06F13/10
EC Classification:
Equivalents:

Abstract

PROBLEM TO BE SOLVED: To more flexibly and easily provide a customizable control system and constructing method therefor, in the control system composed of plural common objects.
SOLUTION: A first OCX (object liking and embedding custom control) 10 to be the instance of a common object is provided with an interface 12 transmitting an event and an interface 13 receiving the event, in addition to an interface providing property and methods. The address of the interface 13 of the first OCX 10 is passed to the interface 12 of a second OCX 11 by an interface object 15 and the connection on the event is formed. Thus, the bi-directional communication between the OCX 10 and 11 becomes possible, the function of the second OCX is made the full use from the first OCX 10 and a system which is capable of obtaining a quick response can be constructed.

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平9－106355

(43)公開日　平成９年(1997)４月22日

(51)Int.Cl.⁶		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	9/46	３４０		Ｇ０６Ｆ	9/46	３４０Ａ
	9/06	５３０			9/06	５３０Ｗ
	9/44	５３０			9/44	５３０Ｍ
	13/10	３３０			13/10	３３０Ａ

審査請求　未請求　請求項の数９　ＯＬ　（全 16 頁）

(21)出願番号　特願平7－264193

(22)出願日　平成７年(1995)10月12日

(71)出願人　000002369
セイコーエプソン株式会社
東京都新宿区西新宿２丁目４番１号

(72)発明者　木村　芳弘
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

(72)発明者　久松　豊
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

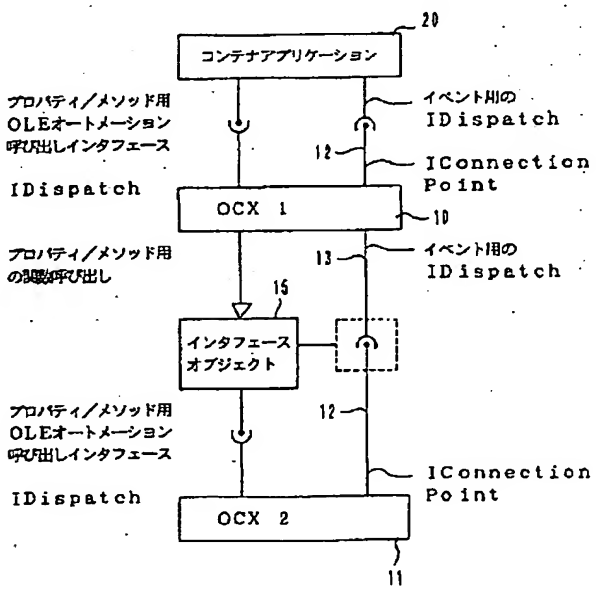
(74)代理人　弁理士　鈴木　喜三郎　（外１名）

(54)【発明の名称】　複数のオブジェクトを用いた制御システム、その構築方法および周辺装置制御システム

(57)【要約】

【課題】　複数のコモンオブジェクトから構成される制御システムにおいて、よりフレキシブルに、また、容易にカスタマイズ可能な制御システムおよびその構築方法を提供する。

【解決手段】　コモンオブジェクトのインスタンスである第１のＯＣＸ１０にプロパティおよびメソッドを提供するインタフェースに加え、イベントを発信するインタフェース１２とイベントを受信するインタフェース１３とを設ける。そして、インタフェースオブジェクト１５によって第１のＯＣＸ１０のインタフェース１３のアドレスを第２のＯＣＸ１１のインタフェース１２に渡し、イベントに関するコネクションを成立させる。これによって、ＯＣＸ１０および１１の間で双方向の通信が可能となり、第１のＯＣＸ１０から第２のＯＣＸの機能をフルに活用し、迅速な応答の得られるシステムを構築できる。



1

【特許請求の範囲】

【請求項１】　属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかを提供する機能と、非同期に発生したアクションを含むイベントを発信する第１のインタフェースとを備えた複数の制御オブジェクトを有し、

第１の前記制御オブジェクトまたはそのインスタンスによって第２の前記制御オブジェクトのインスタンスが作成され、前記第１の制御オブジェクトまたそのインスタンスから前記第２の制御オブジェクトのインスタンスの制御が行われるコンピュータ内のシステムであって、

前記第１の制御オブジェクトは、前記イベントを受信する第２のインタフェースを備えており、さらに、

前記プロパティおよびメソッドの少なくともいずれかを前記第１の制御オブジェクトまたはそのインスタンスと前記第２の制御オブジェクトのインスタンスの間で伝達する機能と、前記第１の制御オブジェクトまたはそのインスタンスの前記第２のインタフェースおよび前記第２の制御オブジェクトのインスタンスの前記第１のインタフェースの少なくともいずれか一方の識別子を他方に渡す機能とを備えたインタフェースオブジェクトまたはそのインスタンスを有することを特徴とする複数のオブジェクトを用いた制御システム。

【請求項２】　請求項１において、前記第２の制御オブジェクトは複数の前記イベントを第２の配列に従って発信し、前記第１の制御オブジェクトは受信した前記複数のイベントの配列を第１の配列に変換する手段を備えていることを特徴とする複数のオブジェクトを用いた制御システム。

【請求項３】　属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかを提供する機能と、非同期に発生するアクションを含むイベントを発信する第１のインタフェースとを備えた複数の制御オブジェクトを有し、第１の前記制御オブジェクトまたはそのインスタンスによって第２の前記制御オブジェクトのインスタンスが作成され、

前記第１の制御オブジェクトは前記イベントを受信する第２のインタフェースを備えており、さらに、

前記プロパティおよびメソッドの少なくともいずれかを前記第１の制御オブジェクトまたはそのインスタンスと前記第２の制御オブジェクトのインスタンスとの間で伝達する機能と、前記第１の制御オブジェクトまたはそのインスタンスの前記第２のインタフェースおよび前記第２の制御オブジェクトのインスタンスの前記第１のインタフェースの少なくともいずれか一方の識別子を他方に渡す機能とを備えたインタフェースオブジェクトまたはそのインスタンスを有する制御システムの構築方法であって、

前記第１の制御オブジェクトまたはそのインスタンスが前記インタフェースオブジェクトのインスタンスを作成

2

するステップと、

前記インタフェースオブジェクトのインスタンスが前記第２の制御オブジェクトのインスタンスを作成するステップと、

前記第１の制御オブジェクトまたはそのインスタンスの前記第２のインタフェースと、前記第２の制御オブジェクトのインスタンスの前記第１のインタフェースとのコネクションを張るステップとを有することを特徴とする制御システムの構築方法。

【請求項４】　請求項３において、前記第１の制御オブジェクトは複数の前記イベントを受信する第１の配列を備えており、前記第２の制御オブジェクトは前記複数のイベントを発信する第２の配列を備えており、前記コネクションを張るステップにおいて、前記第２の配列を前記第１の配列に変換する第３の配列を作成することを特徴とする制御システムの構築方法。

【請求項５】　アプリケーションシステムに対し周辺装置の管理を行う制御システムであって、

属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかを提供する機能と、非同期に発生するアクションを含むイベントを発信する第１のインタフェースとを備えた複数の制御オブジェクトまたはそのインスタンスを有し、

これら複数の制御オブジェクトまたはそのインスタンスからなる制御オブジェクト群は、前記イベントを受信する第２のインタフェースをさらに備えた第１の前記制御オブジェクトまたはそのインスタンスと、

これら第１の制御オブジェクトまたはそのインスタンスによって作成された第２の前記制御オブジェクトのインスタンスと、

前記プロパティおよびメソッドの少なくともいずれかを前記第１の制御オブジェクトまたはそのインスタンスと前記第２の制御オブジェクトのインスタンスの間で伝達する機能と、前記第１の制御オブジェクトまたはそのインスタンスの前記第２のインタフェースおよび前記第２の制御オブジェクトのインスタンスの前記第１のインタフェースの少なくともいずれか一方の識別子を他方に渡す機能とを備えたインタフェースオブジェクトまたはそのインスタンスとを有していることを特徴とする周辺装置の制御システム。

【請求項６】　請求項５において、前記第２の制御オブジェクトは、前記周辺装置の第１の周辺装置に対応していることを特徴とする周辺装置の制御システム。

【請求項７】　請求項６において、前記第２の制御オブジェクトのプロパティは前記第１の周辺装置に固有な仕様に対応した属性値を含み、前記メソッドは前記第１の周辺装置に固有な命令を含み、さらに、前記イベントは前記第１の周辺装置において非同期に発生するアクションを反映可能であることを特徴とする周辺装置の制御システム。

【請求項８】　請求項６において、前記第１の制御オブジェクトは、前記インタフェースオブジェクトを介して伝達された前記第２の制御オブジェクトのプロパティ、メソッドおよびイベントの少なくともいずれかを普遍的な仕様に変換することを特徴とする周辺装置の制御システム。

【請求項９】　請求項６において、前記アプリケーションシステムはＰＯＳ制御システムであり、前記第２の制御オブジェクトはＰＯＳを構成可能な各種の外部周辺装置毎に用意されており、前記ＰＯＳ制御システムは前記第１の制御オブジェクトまたはそのインスタンスを介して前記外部周辺装置を制御することを特徴とする周辺装置の制御システム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、共通に使用できる複数のオブジェクトを備え、これらのオブジェクトを用いてカスタマイズ可能な制御システム、その構築方法および周辺装置制御システムに関するものである。

【０００２】

【発明が解決しようとする課題】近年、コンピュータを用いて多種多様の処理が行われ、コンピュータに接続可能な外部入出力機器も多種多様なものが用意されている。従って、コンピュータを中心として、ユーザーの環境や目的などに合わせてフレキシブルに構築でき、カスタマイズ可能なシステムを提供することが重要となっている。例えば、共通な規格のバスを用いて１つのコンピュータに様々なタイプのプリンター、ディスプレイ、バーコードリーダーを始めその他の入出力機器が接続可能であり、コンピュータにはこれらの入出力機器を制御可能なソフトウェアがインストールされる場合には、これらの機器を的確に制御するように制御システムが構築されていなければならない。また、複数のコンピュータをネットワークで結んだ環境が簡単に構築できるので、個々のコンピュータによって処理すべき対象およびその手順を様々に変更できることが望ましい。

【０００３】このようなフレキシブルなシステムをコンピュータ内に構築する１つの方法として、共通に用いることのできる複数のオブジェクト（コモンオブジェクト）を作成し、これらを活用してオペレーティングシステムやアプリケーションプログラムを構築することが考えられる。マイクロソフト社のＯＬＥ（ＯｂｊｅｃｔＬｉｎｋｉｎｇ　ａｎｄ　Ｅｍｂｅｄｄｉｎｇ）およびＯＬＥプログラミング用に用意されたインタフェースを用いたＯＬＥオートメーションあるいＯＬＥカスタムコントロール（ＯＣＸ）といったシステムはその１つである。

【０００４】本発明においては、コモンオブジェクトを活用して多種多様なオペレーティングシステムあるいはアプリケーションプログラムを簡単に構築できるシステムおよびその方法を提供することを目的としている。特に、１つのコモンオブジェクトが他のコモンオブジェクトの機能を充分に引き出し得る制御システムおよびその構築方法を提供することを目的としている。さらに、多種多様な周辺装置の制御などに適したカスタマイズの容易なオープンなシステムでありながら処理速度の早い制御システムを提供することも本発明の目的の１つである。

【０００５】

【課題を解決するための手段】本発明は、コモンオブジェクトとしての機能を備えた制御オブジェクト、すなわち、属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかをアプリケーションプログラムや他の制御オブジェクトあるいはそのインスタンスに提供する機能と、非同期に発生するアクションを含むイベントを発信する第１のインタフェースとを備えた制御オブジェクトに、さらに、イベントを受信する第２のインタフェースを設け、複数のこれら制御オブジェクトを用いて制御システムを構成するようにしている。

【０００６】従って、本発明の制御システムでは、第１の制御オブジェクトまたはそのインスタンスはイベントを受信する第２のインタフェースを備えており、第２の制御オブジェクトのインスタンスにおいて非同期に発生するイベントを第１のインタフェースおよび第２のインタフェースを介して第１の制御オブジェクトまたはそのインスタンス、あるいは上流のアプリケーションプログラムに遅滞なく伝達できる。このため、第１の制御オブジェクトまたはそのインスタンスによって第２の制御オブジェクトのインスタンスを作成し、それらの制御を行うことによって多種多様な制御システムを簡単に構築でき、処理速度の早い多機能の制御システムとすることができる。

【０００７】第１の制御オブジェクトまたはそのインスタンスの第２のインタフェースと、第２の制御オブジェクトのインスタンスの第１のインタフェースとの間にコネクションを張るためには、少なくともいずれか一方の識別子を他方に渡すことが必要である。本発明においては、プロパティおよびメソッドの少なくともいずれかを第１の制御オブジェクトまたはそのインスタンスと第２の制御オブジェクトのインスタンスの間で伝える機能と、上記の識別子を渡す機能とを備えたインタフェースオブジェクトを設けるようにしている。従って、本発明により、１つの制御オブジェクトまたはそのインスタンスと他の制御オブジェクトまたはそのインスタンスとの間でプロパティ、メソッドおよびイベントの全てを含んだ双方向の通信が可能となる。このため、制御オブジェクトとして提供された機能を充分に活用した制御システムを簡単に構築できる。

【０００８】さらに、複数のイベントが用意されている

場合に、第１の制御システムのイベントの第１の配列と、第２の制御システムの第２の配列とが一致しなくてもイベントの受渡しが可能なように、これらの配列を変換する手段を設けておくことが望ましい。

【０００９】本発明に係る複数の制御オブジェクトおよびそれらのインスタンスを備えた制御システムにより、コンピューターの機種やそれに接続される周辺装置に対しオープンな制御システムを構築できる。例えば、第２の制御オブジェクトを複数の周辺装置の中の第１の周辺装置に対応して用意することによって、第２の制御オブジェクトに第１の周辺装置に固有な仕様に対応した属性値を含んだプロパティと、命令を含んだメソッドと、さらに、第１の周辺装置において非同期に発生するアクションを反映可能なイベントを設定することができる。そして、第２の制御オブジェクトのインスタンスと第１の制御オブジェクトあるいはそのインスタンスをインタフェースオブジェクトあるいはそのインスタンスを介して接続できる。従って、第１の制御オブジェクトあるいはそのインスタンスにおいて第２の制御オブジェクトのインスタンスのプロパティ、メソッドおよびイベントの少なくともいずれかを複数の機種、メーカーあるいは複数の種類の周辺装置に対して普遍的に規定された仕様に変換できる。このため、周辺装置に依存しない共通のアプリケーションプログラムインタフェース（ＡＰＩ）をアプリケーションプログラムや上位の制御オブジェクトおよびそのインスタンスに対し提供することができる。さらに、第１の制御オブジェクトのＡＰＩをコンピュータの機種に依存しない普遍的なものにすることも可能であり、機種依存性がなく、汎用性の高いアプリケーションプログラムとすることができる。従って、本発明の制御システムにより、ユーザーは自己の環境に合わせたアプリケーションおよびハードウェアによってＰＯＳシステムなどの周辺装置を多く使用するシステムを簡単に、そしてフレキシブルに構築できる。

【００１０】

【発明の実施の形態】

〔ＯＬＥオートメーションをベースにした例〕以下において、マイクロソフト社の提供するプログラム開発環境であるMicrosoft Foundation Class（ＭＦＣ）環境下において本発明を実施した例に基づきさらに詳しく説明する。

【００１１】ＭＦＣはＯＬＥのプログラミングを容易に行えるようにさまざまなライブラリーを提供しており、オブジェクト指向のプログラミング言語であるMicrosoft 社が提供するVisual C++（以下VC++）などによってＯＬＥオートメーション機能を用いたシステムの開発を行うことができる。本例では、図１に示すように、アプリケーション（コンテナアプリ）２０と、システム内に共通に活用できるように用意されたコモンオブジェクトあるいはそのインスタンス（ＯＣＸ）１０および１１とを階層的に結合して構築された制御システムについて先ず説明する。

【００１２】コモンオブジェクトは、他の実行型のソフトウェアを構成するオブジェクトとして提供されても良く、あるいはダイナミックリンクライブラリ（ＤＬＬ）として提供されるオブジェクトであっても良い。コモンオブジェクトを提供するこれらの実行型のソフトウェアやＤＬＬはＥＸＥサーバーあるいはＤＬＬサーバーと呼ぶことができ、これらに対し、コンテナアプリ２０をクライアントあるいはコントロールと呼ぶことができる。ＯＬＥに対応したこれらのコモンオブジェクトは、例えば、ＭＦＣが提供しているＣＣｍｄＴａｒｇｅｔクラスから派生させることができる。ＣＣｍｄＴａｒｇｅｔクラスは、コモンオブジェクトを管理するためのＩＵｎｋｎｏｗｎインタフェースや、その他のコモンオブジェクトとして必要なインタフェース等をサポートするクラスである。ＣＣｍｄＴａｒｇｅｔクラスはコントローラ側からデータを保持した構造体などを受け渡すためのIDispatch インタフェースもサポートしており、このメンバ関数Invokeをコールすることによってコモンオブジェクト側のプロパティあるいはメソッドを用いることができる。

【００１３】プロパティは、コモンオブジェクトの属性であり、カラー、テキスト、番号、フォントあるいはプッシュボタンがプッシュされた時の動作などが含まれる。メソッドは、コモンオブジェクトにインプリメントされた、例えば編集機能などのファンクションであり、メソッドをコールすることによってコモンオブジェクトの機能を操作することができる。コモンオブジェクトは、プロパティやメソッドを提供する機能に加え、コモンオブジェクトに対する外部からのアクション、例えばマウスボタンのクリックやキー入力、その他の非同期に発生するアクションをイベントとして発信する機能を備えることができる。これは通常、コントロール側がプロパティー等を用いるのと同様に、コモンオブジェクトにおいてInvoke関数をコールすることによって行われる。このとき、イベント用の結合点が有効なインタフェースを保持していないとイベントの発生は無視されてしまう。イベントの結合点を実現するインタフェースはIConnectionPoint１２である。

【００１４】さらに、本発明においては、コモンオブジェクトにイベントを受けるためのインタフェースとしてイベント用のIDispatch １３を設けてある。従って、ＯＣＸの間でイベントの授受を行う場合は、イベントを発信するＯＣＸのインタフェースはIConnectionPoint１２に対し、イベントを処理するＯＣＸのイベント用のIDispatch １３をアドバイスする、すなわちアドレス等の識別子を渡し、コネクションを張れば良い。もちろん、ＭＦＣの環境下以外においては、その他の識別子であっても良く、また、インタフェース１２または１３のいずれ

に側に対して他方の識別子を渡しても良い。

【００１５】図１には、コンテナアプリ２０があるコモンオブジェクトあるいはそのインスタンスを用いている制御システムを示してある。本図に示したシステムでは、コンテナアプリ２０が第１のオブジェクトのインスタンス（ＯＣＸ）１０を作成し、さらに、その第１のＯＣＸ１０が別の第２のオブジェクトのインスタンス（ＯＣＸ）１１を作成して制御する制御システムを示してある。ＭＦＣライブラリーには、コンテナアプリやＯＣＸ等からInvoke関数をコールする複雑さをカバーするためにCOleDispatchDriverクラスが用意されており、ＯＣＸ１０はこのクラスから派生させたオブジェクトのメンバー関数をコールすることにより、他の制御オブジェクトのインスタンス、例えば、ＯＣＸ１１を作成しＯＣＸ１１のメソッド等を活用することができる。

【００１６】サーバー内のコモンオブジェクトを用いる場合、例えば、コンテナアプリの動作しているプロセッサーと異なるプロセッサーでそのコモンオブジェクトが動作しているケースも考えられる。このようなケースでは、本例のような制御システムを、コントロール側とサーバー側で通信処理を行うことにより構築することができる。また、１つのプロセッサーに活用するコモンオブジェクトをコピーして制御システムを構築することも可能である。いずれの場合も、用いられるコモンオブジェクトのコード自体には変更は全く加えられず、そのコモンオブジェクトを活性化する毎にデータやスタックなどの領域が用意され、それを用いてコモンオブジェクト（サーバー側）とコントロール側が通信しながら制御システムを構築する。従って、同一のコモンオブジェクトをコントロール側で複数用いることも可能であり、この場合、コモンオブジェクトが複数コピーされるのではなく、データやスタックなどの領域がそれぞれに用意される。このようにコモンオブジェクトが他のオブジェクト等によって活用可能な状態になることを本明細書ではインスタンスと称する。

【００１７】さて、第１のＯＣＸ１０がコントロール側となって第２のＯＣＸ１１を呼び出して使用したい場合、VC++のクラスウィザードを用いてCOleDispatchDriverクラスから派生されたインタフェースオブジェクトを用いることが可能であることは上述した通りである。なお、本明細書においては、インタフェースオブジェクトとは、インタフェースオブジェクトおよびそのインスタンスを含む概念で用いている。VC++2.0 のCOleDispatchDriverクラスは、メソッドやプロパティをIDispatch インタフェースを介して提供する機能はサポートしているが、第２のＯＣＸが発信するイベントをとらえる機能をサポートしていない。従って、コンテナアプリ２０の側が第１のＯＣＸ１０の用いる第２のＯＣＸ１１を識別し、第２のＯＣＸ１１のイベントを捕らえる機能をサポートしている場合などの特殊なケースを除いて第２のＯＣＸ１１で発生したイベントをとらえることができない。

【００１８】本例においては、ＯＣＸ間の双方向通信を可能とするためにＯＣＸにイベント用のインタフェース１３を設けると共に、COleDispatchDriverクラスからイベント用のインタフェースを結合できるインタフェースオブジェクトを派生させている。すなわち、本例の制御システムは、上述したイベントを発信するインタフェースであるIConnectionPoint１２とイベントを受信するインタフェースであるイベント用のIDispatch １３を備えたＯＣＸ１０と、その下流に位置するイベントを発信するインタフェースであるIConnectionPoint１２を備えたＯＣＸ１１を有しており、これらをインタフェースオブジェクト１５により接続している。本例のインタフェースオブジェクト１５は、ＯＣＸ１１のプロパティおよびイベントをＯＣＸ１０に提供する機能に加え、ＯＣＸ１．１のIConnectionPoint１２にＯＣＸ１０のイベント用のIDispatch １３のアドレスを渡す機能を備えるようにしている。このため、ＯＣＸ１０が本例のインタフェースオブジェクト１５をインスタンスとして作成し、さらに、インタフェースオブジェクト１５がＯＣＸ１０に基づきＯＣＸ１１を作成すると、ＯＣＸ１１のIConnectionPoint１２とＯＣＸ１０のイベント用のIDispatch １３の間にコネクションが張られ、ＯＣＸ１１の発信したイベントがＯＣＸ１０によって受けれるようになる。従って、ＯＣＸ１１とＯＣＸ１０との間で双方向の通信が可能となり、コンテナアプリ２０の側は第２のＯＣＸ１１を認識する必要は全くなくなる。また、第２のＯＣＸ１１もコンテナアプリ２０に左右されないオブジェクトとして提供することができる。

【００１９】本例においては、インタフェースオブジェクト１５は、COcxDispatchDriverクラスから派生されている。このCOcxDispatchDriverは、CCmdTargetクラスと、COleDispatchDriverクラスとを多重継承(Multiple Inheritance)して派生したオブジェクトクラスであり、次のようになる。

【００２０】

```
class COcxDispatchDriver:public CCmdTarget,public COleDispatchDriver
{ .... } ;
```

CCmdTargetクラスは上述したように、サーバーとしての機能を備えたオブジェクトを派生するクラスであり、また、COleDispatchDriverクラスは、第２のＯＣＸに対してクライアントあるいはコントロール側として動作し、プロパティ／メソッドに対するアクセスを簡単に行なえる機能を備えたオブジェクトクラスである。従って、クラスウィザードにより上記の２つのクラスから派生させたオブジェクトクラスであるCOcxDispatchDriverクラス

は、上記の２つの主な機能を備え、これらに加えそれぞれのクラスの機能を全て継承し、サポートされたオブジェクトクラスなので、本発明に係るインタフェースオブジェクト用のオブジェクトクラスとして適している。CCmdTargetクラスとCOleDispatchDriverクラスには重複す＊る部分がないため、全く問題なく多重継承が可能である。

【００２１】このように派生されたCOcxDispatchDriverクラスの仕様は以下の通りである。

【００２２】

```
Class COcxDispatchDriver : public CCmdTarget,
                           public COleDispatchDriver
COcxDisptchDriver::COcxDispatchDriver()  //  生成
public:  //  メンバ変数
IID  m __IIDEvents;  //    イベント  インタフェース ID
UNIT m __nEvents;    //    イベント数
CDWordArray m__dispID  //  ユーザが定義したディスパッチマップに対応して
                          ディスパッチIDを変換するための配列
public:  // メンバ関数
Bool PrecreateDispatch(
          REFCLISD clsid,
          CStringArray& EntryNames,
          CDWordArray& DispIDs,
          CSstringArray& ParamInfo,
          COleException* pError=NULL); //    オブジェクトインスタンス
                          を作成し、インタフェースの情報を格納する。
```

【００２３】

```
Bool EstablishConnection(
          LPUNKNOWN pUnkSink,
          LPCDMENTRY pDispMap,
          CStringArray& EntryNames,
          CDWordArray& DispIDs,
          CStringArray& ParamInfo,
          COleException* pError=NULL); //    イベントのインタフェース
                          が受側と発行する側とで一致しているかを確認し
                          双方を結合する。
```

【００２４】

```
void DestroyConnection(); //  コネクションを開放し、オブジェクト
                             インスタンスを削除する
```

本例のCOcxDispatchDriverクラスから派生されるインタフェースオブジェクトは、ＯＣＸ双方のイベントのインタフェースのコネクションを張る機能を備えているだけなのでイベントの処理関数（イベントハンドラ）は含んでいない。イベントハンドラは、イベントを受け入れるＯＣＸの側に用意される。まず、イベントを発行する側のＯＣＸが返すイベントの外部名に合わせてディスパッチマップを記述する。ディスパッチマップの各イベントの順番は任意で良く、後述するようにイベントとのコネ※クションをセットするときに実際に受信するイベントの順番との対応付けがなされる。ただし、ＯＣＸの全部のイベントを完全に記述しておく必要がある。

【００２５】ディスパッチマップのエントリは例えば以下のようになり、マクロのパラメータは、順に、ディスパッチドライバクラス名、外部イベント名、イベントハンドラ名、戻り値、パラメータ情報となる。

【００２６】

```
DISP __FUNCTION( __DSoprn, "ControlCompleteEvent", ControlCompleteEvent

VT __EMPTY, VTS__I4 VTS__SCODE VTS __PBSTR VTS __I4)
```

このような外部イベント名に合わせたディスパッチマップを記述することにより、ＯＣＸで発生したイベントを直観的に判りやすい形式で受け取ることができる。さらに、イベントハンドラのプロトタイプ宣言は下記のようになるので、通常の関数の形式で記述でき、プログラム開発が極めて容易となる。

【００２７】

void __DSoprn::ControlCompleteEvent(long ControlID, SCODE Result,
BSTR FAR*, pString, long data);

次に、COcxDispatchDriverクラスから派生されたインタフェースオブジェクト１５によって第１のＯＣＸ１０と第２のＯＣＸ１１とのコネクションを設定する処理を図２および図３に示したフローチャートに基づき説明する。

【００２８】図２に示すように、インタフェースオブジェクトのインスタンス１５が生成され、PrecreateDispatch がコールされると、ステップ３１において、clsid によって指定された第２のＯＣＸであるオブジェクトインスタンス１１が生成され、動作を開始する。変数clsid は作成するＯＣＸを識別するための１２８ビットの値が格納されている。ステップ３２において第２のＯＣＸ１１が動作を開始すると、ステップ３３においてこのＯＣＸ１１のIDispatch インタフェースを取得し保存する。インタフェースオブジェクト１５は、このIDispatch を通じてInvoke関数をコールし、インスタンス１１にプロパティまたはメソッドを渡すことができる。ステップ３４において、第２のＯＣＸ１１にプロパティあるいはイベントを渡すインタフェースが確認されると、ステップ３５において、第２のＯＣＸ１１のイベントのインタフェースがチェックされる。

【００２９】ステップ３６において、イベントに関するインタフェースがあるとステップ３７において、イベントに関する情報を格納する。第２のＯＣＸ１１において用意されているイベント名リストはEntryNamesに、ディスパッチＩＤリストはDispIDs に、また、パラメータ情報リストはParamInfo に格納される。ステップ３８においてこれらの処理が通常に終了すると次に図３に示したイベントの対応付けを行う工程に移行する。一方、コネクションがセットできない場合はステップ３９においてエラー処理を行う。

【００３０】図３にイベントの対応付けを行う工程を示してある。pUnkSinkにイベントの受入れ側となるＯＣＸ１０のインタフェースの識別子（本例ではＯＣＸ１０のイベント用IDispatch のアドレス）が設定され、EstablishConnection がコールされる。まず、ステップ４１においてカウンタfcntをゼロクリアーする。次にステップ４２においてカウンターfcntを確認し、ステップ４３において第２のＯＣＸ１１から得られたイベント名EntryNames(fcnt)がＯＣＸ１０のイベントハンドラに予め用意したマップpDispMapにあるか否かをチェックする。マップpDispMapにある場合は、さらに、ステップ４４において、パラメータ情報ParamInfo (fcnt)がイベントハンドラに予め用意したマップpDispMapと一致するかを確認する。確認できた場合は、ステップ４５において、第２のＯＣＸ１１から得られたイベントのディスパッチＩＤであるDispIDs (fcnt)をイベントハンドラに予め用意したマップpDispMapでのインデックスに変換するための配列m __dispIDを設定する。ステップ４６において、カウンタfcntをインクリメントし、すべてのイベントに対し、これらのステップを繰り返す。予め用意したマップpDispMapとの対応が付くと、ステップ４７において、pUnkSinkにある第１のＯＣＸ１０のインタフェースの識別子を第２のＯＣＸ１１のイベントに関するインタフェースに渡しコネクションを張る。本例では、ＭＦＣ環境下にあるので、IConnectionPointにアドレスを渡し、イベントに関するコネクションをセットする。これによって、第２のＯＣＸ１１においてInvoke関数をコールすると、イベントがＯＣＸ１０に渡される。すなわち、第１のＯＣＸ１０が第２のＯＣＸ１１からイベントを受け取れるようになる。

【００３１】ステップ４８においてコネクションが張られると、ステップ４９においてイベントのディスパッチＩＤを変換するための配列m __dispIDを保存する。ステップ５０において、上記のステップが全て正常に終了するとインタフェースオブジェクト１５と第２のＯＣＸ１１とのコネクションのセットが終了し、第１のＯＣＸ１０に対し第２のＯＣＸ１１のプロパティあるいはメソッドを提供できるようになり、さらに、第２のＯＣＸ１１から第１のＯＣＸ１０にイベントが伝えられるようになる。

【００３２】このように、コモンオブジェクトにイベントを受け取るためのインタフェースを用意し、さらに、インタフェースオブジェクト１５によってインタフェース間のコネクションを張れば、コモンオブジェクトあるいはそのインスタンス同士の間で双方向の通信が可能となる。従って、コモンオブジェクトあるいはそのインスタンスを用いて処理の速いフレキシブルな制御システムを構築することができる。さらに、本例のインタフェースオブジェクトにおいては、イベントに関するコネクションをセットするときに、イベントを発信する側のイベントの配列と、イベントを受け入れる側のイベントの配列を確認し、対応付けるようにしている。従って、第１のＯＣＸ１０に対し、第２のＯＣＸ１１のインタフェースが完全に既知となっていなくとも、イベントの種類と数が合致していればコネクションをセットすることができる。すなわち、プロパティ、メソッドおよびイベントの名称が合致していれば複数のコモンオブジェクト間に双方向のコネクションをセットすることができる。このため、コモンオブジェクトの開発が容易となり、バージョンアップが行われたり、異なったオブジェクトが採用された場合であっても、制御システムをフレキシブルに、そして、安全、確実に構築することができる。

【００３３】さらに、イベントを受け取ったときに起動されるイベントハンドラも通常の関数の形式で記述することができるので、オブジェクトの開発が極めて容易と

なり短時間で行えるようになる。

【００３４】図４に、第２のＯＣＸ１１から、インタフェースオブジェクト１５に用意されたIDispatch インタフェースのイベントを通知する呼び出し Invoke(EventID,....）がコールされた場合の処理を示してある。IDispatch インタフェースは Invoke() および、それをサポートする基本的な関数（AddRef(), Release(),QueryInterface(IID&, LPUNKNOWN*) 等）を提供しており、IDispatch インタフェースが、相手のＯＣＸが発生するイベントの IID（Interface ID）にQueryInterface(...）を介して応答する。 Invoke() がコールされると、この時点でパラメータとしてイベントの種類を規定するdispatch ID やイベントのパラメータを保持する構造体などがＯＣＸ１０に渡される。イベントを受け入れる側のＯＣＸ１０では、ステップ６０においてパラメータが正当か否かをチェックする。正当の場合はステップ６１において、第２のＯＣＸ１１から得られたイベントの配列dispIDMemberを用意されたイベントハンドラのマップpDsipMapに合わせて、変換用の配列m __dispIDに基づき変換する。ステップ６２において、dispIDMemberに対応するイベントハンドラをpDsipMapから取り出す。そして、ステップ６３において、イベントハンドラが有効か否かを確認し、有効な場合はステップ６４においてイベントに対応するイベントハンドラを呼び出し、イベントハンドラに定義された処理、例えば、アプリケーション側にイベントを発信するなどの処理を行う。

【００３５】なお、上記では、ＯＣＸ１０に１つのＯＣＸ１１が繋がる場合を例に説明しているが、複数のＯＣＸ１１と繋げることももちろん可能である。イベントを受け入れるインタフェースが単独の場合は、同一のイベントの配列である必要があるので、ＯＣＸ１１のタイプは同一であることが望ましい。

【００３６】このように、本発明のシステムにおいては、複数のコモンオブジェクトおよびそれらのインスタンス、そしてインタフェースオブジェクトを用いることにより、フレキシブルな制御システムを簡単に構築することができる。それぞれのコモンオブジェクトには、活用する相手のコモンオブジェクトに関し最小限の情報を用意しておけば良く、また、活用される相手のコモンオブジェクトに関する情報も最小限で済む。従って、個々のコモンオブジェクトのカプセル化をいっそう進めることができる。その一方で、本発明のシステムでは、ＯＣＸ同士の間でイベントも含めた双方向の通信が可能であり、活用する相手のＯＣＸを完全に動作させることができ、それらの備えた機能を最大限に発揮させることができる。

【００３７】なお、本例においては、第１および第２のＯＣＸからなる２層の制御システムを用いて説明しているが、３層あるいはそれ以上の階層構造をもった制御システムであっても同様に構築することができる。また、上記においては、マイクロソフト社の提供するＭＦＣ環境下において本発明のオブジェクトインタフェースを実現した例を用いて説明しているが、ＭＦＣライブラリを用いなくとも、同等の機能を備えたオブジェクトインタフェースを作成することは可能である。さらに、共通して活用できるオブジェクトが用意されたシステム環境であれば、マイクロソフト社のＯＬＥオートメーションでなくとも上記と同様のコモンオブジェクトを用いたシステムを構築できる。このようなコモンオブジェクトを用いたシステムは、１つのプロセッサーを備えたコンピュータ単体で実現することもでき、また、ネットワーク環境で接続された複数のプロセッサーを有するシステムにおいても構築できることは上述した通りである。

【００３８】また、本例では、インタフェースオブジェクトを採用することによって、より拡張性が高く、また、プログラム開発の容易な制御システムをしているが、インタフェースオブジェクトの機能と第１のＯＣＸの機能を備えたコモンオブジェクトを提供することももちろん可能であり、そのようなＯＣＸを用いたシステムによってもイベントを含めた双方向通信の可能な制御システムを構築できることはもちろんである。

【００３９】〔周辺装置の制御システム〕図５に、本発明に係るインタフェースオブジェクトと、複数のコモンオブジェクトおよびそのインスタンスによって周辺装置の制御システムを構築した例を示してある。図５にはパーソナルコンピュータ（パソコン）７０を中心に構成されたＰＯＳシステムを示してある。パソコン７０にはＰＯＳアプリケーションプログラム７１がインストールされており、パソコンのオペレーティングシステム（ＯＳ）１０５の上で動作する。ＯＳ１０５は、キーボードドライバ１０６やモニタディスプレイドライバ１０７等を介してパソコンとして通常必要な周辺装置を制御する機能を備えており、アプリケーションプログラム７１とキーボードあるいはモニタディスプレイ（不図示）との間のデータ転送はＯＳ１０５を介して行われる。

【００４０】ＰＯＳシステムには、パソコンに通常用意されるこれらの周辺機器に加えて、客先に金額等を表示するカスタマディスプレイ１１０、レシート等の印刷を行うレシートプリンタ１１２、チェックなどの印刷を行うスリッププリンタ１１３、さらに、金銭を保管するキャッシュドロワ１１５が必要となり、これらの周辺装置はＲＳ－２３２Ｃポート等の拡張用のポートに接続される。例えば、カスタマディスプレイ１１０がＲＳ－２３２Ｃポートに接続され、カスタマディスプレイ１１０をパススルーしてレシートプリンタ１１２およびスリッププリンタ１１３を備えたプリンタ１１１が接続される。キャッシュドロワ１１５は、プリンタ１１１の下部に設置され、プリンタ１１１の制御機構を介して操作される。これらの周辺装置は多くのメーカーから様々な機種が市販されており、ユーザーは自己の環境に適したもの

を選択してＰＯＳシステムを構築可能である。しかしながら、メーカーや機種の異なる周辺装置は個々に仕様が異なるので、市販されている全ての周辺装置に適合したアプリケーションプログラムの作成は不可能である。さらに、周辺装置がバージョンアップされると、これに伴い仕様も変更になる。従って、従来は、ユーザーは自己に都合の良い周辺機器によってＰＯＳシステムを構築することが困難なこともあり、さらに、周辺機器がバージョンアップされても新しい機種を構築済のＰＯＳシステムにすぐに適用できるとは限らなかった。

【００４１】これに対し、上述したＯＣＸによって制御システムを構築すると、非常にオープンなシステムを形成できる。従って、どのような機種の周辺機器を用いたＰＯＳシステムであっても簡単に構築できる。また、周辺機器のバージョンアップに対しても簡単に対処できる。例えば、図５に示す本例の周辺機器の制御システム７２は、３つのレベルのＯＣＸを備えている。第１のレベルのＯＣＸとして、レシートプリンタフォーマット変換用ＯＣＸ７３と、スリッププリンタフォーマット変換用ＯＣＸ７４が用意されている。これらのＯＣＸ７３および７４では、例えば、アプリケーションプログラム７１から送られた売上品リストや合計金額等のデータを所定のフォーマットに配置する処理が行われる。所定のフォーマットは、ＯＣＸ７３あるいは７４の内部に設定されていても良いし、下位レベルの制御用ＯＣＸであるレシートプリンタ制御用ＯＣＸ７５あるいはスリッププリンタ制御用ＯＣＸ７６がプロパティとして有しており、これらの制御用ＯＣＸのプロパティを得て変換用ＯＣＸ７３あるいは７４がフォーマットを設定しても良い。いずれにしろ、アプリケーションプログラム７１は、出力表示されるフォーマットに関係なく、出力用のデータを変換用ＯＣＸ７３あるいは７４に受け渡せば良い。従って、インタフェースの形式を限定でき、汎用性の高いアプリケーションプログラムとして提供することができる。また、アプリケーションプログラムから固有のフォーマットで出力されたデータを、第１レベルのＯＣＸによって下位のＯＣＸに共通するフォーマットに変換することも可能である。このように、ＯＣＸを用いて個別に開発されたアプリケーションプログラムの汎用性を高めることも可能である。

【００４２】第２のレベルのＯＣＸとしてレシートプリンタ制御用ＯＣＸ７５、スリッププリンタ制御用ＯＣＸ７６、キャッシュドロワ制御用ＯＣＸ７７およびカスタマディスプレイ制御用ＯＣＸ７８が用いられている。これらのＯＣＸ７５～７８は、アプリケーションプログラムあるいは上位のＯＣＸに対し、所定の仕様のインタフェース（ＡＰＩ）を提供するＯＣＸである。従って、ＰＯＳシステムを構成するプリンタ等の周辺装置のメーカーや機種に係わりなく、アプリケーションプログラムや上位のＯＣＸは所定の仕様でデータを提供すれば良い。

このレベルのＯＣＸ７５～７８は、周辺装置固有の仕様が反映された下位のドライバレベルのＯＣＸのプロパティを得て、共通の仕様で入力されたデータを下位のＯＣＸ、すなわち、実際にシステムを構築している周辺装置の仕様に合わせたデータに変換する。

【００４３】第３のレベルのＯＣＸは、プリンタドライバＯＣＸ９１および９２と、キャッシュドロワドライバＯＣＸ９３、およびディスプレイドライバＯＣＸ９４である。これらのＯＣＸ９１～９４は、個々の周辺装置に対応したコモンオブジェクトのインスタンスであり、通常はメーカー毎あるいは機種毎に異なり周辺装置と共に提供される。これらのドライバレベルのＯＣＸ９１～９４は、例えば、最大印字行数や印字行ピッチと言ったプリンタ固有の仕様や設定状態（プリンタステータスと称する）をプロパティとして備えており、これらのプロパティは上位のＯＣＸやアプリケーションプログラムで参照できるようになっている。また、指定された位置に文字列を印字する、すなわち、印字命令を出力するというメソッドや、プリンタステータス等のプロパティなどを備えている。従って、ドライバーレベルである第３レベルのＣＯＸからは、例えば、プリンタ用のＯＣＸであれば、送られた印字位置および印字文字列のデータに基づき、改行量（すなわち行ピッチ）、改行コマンド、印字データおよび印字コマンド、オートカットコマンドが所定の順序でポートドライバ１００を介してプリンタに送信される。

【００４４】第３レベルのＯＣＸは、プリンタから送信される処理結果およびエラーステータス等の非同期に発生するアクションも受け取る。そして、これらのアクションの内容を対応する周辺装置の仕様に基づき解釈し、普遍的な形式に変換してイベントとして上位のＯＣＸに返す。

【００４５】本例の制御システムにおいては、少なくとも第１および第２のレベルのＯＣＸは上述したイベント用をIDispatch インタフェースを備えたＯＣＸである。従って、本発明に係るインタフェースオブジェクト８１ａ、８１ｂ、８２ａ～８２ｄを用いて上位のＯＣＸと下位のＯＣＸを接続することにより、プロパティやメソッドに加え、イベントを伝達するインタフェースを上位および下位のＯＣＸの結ぶことができる。従って、第２レベルの制御用ＯＣＸ７５～７８は第３レベルのドライバクラスのＯＣＸ９１～９４の発行するイベントを受けて、そのまま、あるいはさらに普遍的な形式に変換し上位のＯＣＸやアプリケーションプログラムに伝達する。これに基づき、アプリケーションシステムや上位のＯＣＸは、ユーザーに対してメッセージを出力したりエラー処理ルーチンを起動するなどの処理を行う。従って、本例の制御システムにおいては、非同期に発生するアクションに対して迅速にそして的確な処理を行うことができる。

【００４６】このように、本例の制御システムは、ＯＣＸを用いているのでユーザーの採用する周辺装置に適合したカスタマイズが容易に行えるシステムであり、また、ＯＣＸ間でプロパティ、メソッドおよびイベントの双方向の通信が確保されているので、処理速度が早く的確に処理が行える制御システムである。

【００４７】レシート印字機能を例にとってさらに詳しく説明する。第３レベルのＯＣＸであるプリンタドライバＯＣＸ９１は、レシートプリンタ１１２のプリンタステータスをプロパティとして備えている。また、プリンタ１１２に印字命令を出力するメソッドを備えている。さらに、プリンタ１１２からリアルタイムで送られてくる用紙なし（ランアウト）やカバーオープンといったエラーステータスに対応して非同期にイベントを発生する。

【００４８】アプリケーションプログラム７１からデータが第１レベルのレシートプリンタフォーマット変換用ＯＣＸ７３に渡され、所定のフォーマットに変換されたのち、第２レベルの制御用ＯＣＸ７４に渡される。そして、制御用ＯＣＸ７４がドライバＯＣＸ９１の印刷を実行するメソッドをコールすると、ドライバＯＣＸ９１がレシートプリンタ１１２のプリンタステータスに合致した適当なコマンドおよびデータをレシートプリンタ１１２に対し送信し、レシート印字を実行する。レシートプリンタ１１２は、送信されたコマンドを実行し、所定のステータスをドライバＯＣＸ９１に返す。ドライバＯＣＸ９１は、返されたステータスを解釈し、エラーステータスがアクティブでなければイベントを発生せず、印刷処理は終了する。また、上位のＯＣＸであるレシートプリンタ制御用ＯＣＸ７５は、プリンタステータスのプロパティを参照することによってプリンタ処理の結果を知ることももちろん可能である。

【００４９】一方、印刷実行のコマンドを実行した結果、あるいは待機中にレシートプリンタ１１２にエラーが発生してエラーステータスがアクティブになると、プリンタ１１２はエラーステータスをドライバＯＣＸ９１に送信する。この場合、ドライバＯＣＸ９１はイベントを発生し、エラーの発生を上位の制御用ＯＣＸ７４に伝達する。制御用ＯＣＸ７４は、イベントに対応して特定の処理を行うことも可能であるし、さらにフォーマット変換用ＯＣＸ７４およびアプリケーションプログラム７１にイベントを通知し、注意を喚起することによって所定の処理を行わせることも可能である。

【００５０】これらのプロパティ、メソッドおよびイベントに加え、スリッププリンタ１１３に対応したドライバＯＣＸ９２としては、スリップ印字に特有のプロパティおよびイベントを設けておくことが望ましい。例えば、プロパティにスリップ用紙の有無を加えることによって、上位のＯＣＸやアプリケーションが印刷実行を指示するタイミングの判断が可能となる。また、イベントに用紙挿入検出、用紙終端検出を加えることにより、アプリケーションプログラムなどにスリップ印刷後の処理やエラー処理をスムーズに行わせることができる。

【００５１】キャッシュドロワ制御用ＯＣＸ７７はキャッシュドロワドライバＯＣＸ９３と通信する。キャッシュドロワドライバＯＣＸ９３はキャッシュドロワオープンをメソッドとして備えており、キャッシュドロワのオープン・クローズを含めたキャッシュドロワステータスをプロパティとして備えている。また、イベントとしてキャッシュドロワエラーやキャッシュドロワオープン検出を発生することができる。

【００５２】カスタマディスプレイ制御用ＯＣＸ７８は、ディスプレイドライバＯＣＸ９４と通信し、このドライバＯＣＸ９４は、表示位置や表示文字列を指定した表示命令をメソッドとして備えている。また、プロパティとしては表示桁数や表示色などのディスプレイ仕様を備えており、ディスプレイ制御用ＯＣＸ７８はこのプロパティに沿って表示データを与える。また、カスタマディスプレイはイベントに該当するエラー等の発生は少ないので、イベントを発生する機能を削除することも可能である。

【００５３】このように、本発明の制御システムは、複数のＯＣＸを用いて階層的な構造を備えており、エラー等を含めた非同期に発生するアクションに対しても迅速に対処可能な制御システムである。また、本例の制御システムは、パソコンに接続される周辺装置に対しフレキシブルに対応できるオープンな制御システムである。例えば、プリンタ１１１は、レシート印字機能１１２と、スリップ印字機能１１３と、さらにドロワ１１５の制御機能とを備えており、制御システムはそれぞれの機能毎にＯＣＸ９１～９３が用意されている。このプリンタ１１１に対し、スリップ印字機能の仕様（例えば、最大印字桁数、実行可能なコマンドセット等）の異なるプリンタに置き換える場合は、スリップ印字機能を制御するドライバ用のコモンオブジェクトを新しいプリンタの仕様に対応するものに置き換えれば良い。周辺装置の制御システムにおいては、置き換えられた新しいドライバ用のコモンオブジェクトからＯＣＸを作成し新しいプリンタに適合した制御システムを自動的に形成する。プリンタがバージョンアップされて仕様が変わった場合でも、同様にドライバ用のコモンオブジェクトを置き換えるだけで良く、その他のコモンオブジェクトやアプリケーションプログラムに変更を加える必要はない。

【００５４】カスタマディスプレイについても同様であり、仕様が変更された場合等はドライバ用のコモンオブジェクトを変更すれば良い。また、バーコードリーダ等の他の周辺装置が加えてＰＯＳシステムが構成される場合は、それに対応した制御用のコモンオブジェクトとドライバ用のコモンオブジェクトがパソコンのシステム内に用意されておれば、アプリケーションあるいは上記の

ＯＣＸによって、下位のＯＣＸが作成され、バーコードリーダ等を含めた周辺装置の制御システムが形成される。

【００５５】同様に、アプリケーションプログラムとのインタフェースを担当する上位のＯＣＸがパソコンのハード等に起因する仕様の違いを吸収し、アプリケーションに対し共通のインタフェースを与えるものであれば、パソコンのハード等の相違とは無関係に共通のアプリケーションプログラムを用いてシステムを構築することも可能となる。パソコン内に用意されたコモンオブジェクトから制御システムがアプリケーションプログラムによって作成され、制御システムを構成するＯＣＸによってパソコンおよびそれに接続された周辺装置に適合したシステムが自動的に形成される。このように、本発明のＯＣＸを用いた制御システムはオープンなシステムであり、インタフェースの仕様を予め合意しておけば、アプリケーションを供給する側は、パソコンやそれに接続される周辺機器の仕様および接続方法等は一切考慮せずにアプリケーションを開発し供給することができる。従って、短期間で開発が可能となり、安価に提供できる。また、ユーザーもパソコンや周辺装置に係わりなく、自己の目的や環境に適したアプリケーションを自由に選択することができる。

【００５６】一方、周辺装置を提供する側も、供給する周辺装置の仕様に対応したコモンオブジェクトを用意し、例えば、周辺装置と共に供給することによって、パソコンやアプリケーションに限定されない汎用的な周辺装置を提供することができる。従って、ユーザーも自己の目的や環境に適した周辺装置を自由に購入し、システムを構築することができる。

【００５７】さらに、本発明に係る制御システムでは、インタフェースオブジェクトによってコモンオブジェクトおよびそのインスタンス同士の間で双方向にプロパティ、メソッドおよびイベントの伝達が可能となっている。従って、本発明の基づく制御システムでは、システムの応答が遅れたり、システムの機能が限定される等のシステムをオープン化する上での弊害は全くない。また、制御システムの機能アップやバージョンアップも個々の制御オブジェクト毎に行うことができるので、システム開発を効率良く、短期間に行える。また、ユーザーもシステムのグレードアップや変更等を簡単に行えるようになる。

【００５８】なお、上記では、用いられる周辺装置の種類や数が多く、また、ユーザーの環境に合わせて様々な機種の周辺装置が用いられる例としてＰＯＳシステムを説明しているが、ＰＯＳシステムに限定されないことはもちろんである。近年のパソコンを中心としたシステムは、ユーザーの目的や能力に合わせて様々な仕様の周辺装置が組み合わされるようになりつつある。本発明に係る共通する制御オブジェクトを用いた制御システムは、様々なユーザーの要望に対してフレキシブルに対応可能なシステムであり、ＰＯＳシステムに限らず、今後様々なシステムに対して適用可能である。

【００５９】

【発明の効果】以上に説明したように、本発明においては、アプリケーションプログラム間で共通に活用できる制御オブジェクト、すなわち、コモンオブジェクトおよびそのインスタンスを用いて制御システムを構築しており、さらに、これらコモンオブジェクトおよびそのインスタンスの間で双方向の通信を可能としている。従って、コモンオブジェクトを活用して多種多様なオペレーティングシステムあるいはアプリケーションプログラムを簡単に構築することができる。このように、本発明は、入出力端末や、処理する対象などにより多種多様に変化するコンピュータを中心とした近年のシステムにおいて、いっそうフレキシブルに、そしてカスタマイズ可能なシステムを実現するのに好適な制御システムおよびその構築方法を提供するものである。

【００６０】特に、上述したようなコンピューターの機種やそれに接続される周辺装置の種類および数量がユーザーの環境に応じて多種多様に変化する制御システムを構築するのに本発明は好適である。本発明に係る制御システムを用いることにより、アプリケーションプログラムに対し周辺装置に依存しない共通のアプリケーションプログラムインタフェースを提供可能である。さらに、本発明に係る制御システムは、汎用性の非常に高い制御システムでありながらエラー等の非同期に発生するアクションに対しても迅速にアプリケーションを作動させることが可能である。従って、ユーザーに対し自己の目的に対し最適な快適な動作環境を容易に構築できる制御システムを提供できる。

【図面の簡単な説明】

【図１】本発明の実施例の制御システムの概略構成を示すブロック図である。

【図２】図１に示す制御システムにおいて、インタフェースオブジェクトとオブジェクトインスタンスとの間でコネクションを確認する過程を示すフローチャートである。

【図３】図１に示す制御システムにおいて、インタフェースオブジェクトとオブジェクトインスタンスとの間でイベントの対応付け、コネクションを確立する過程を示すフローチャートである。

【図４】図１に示す制御システムにおいて、オブジェクトインスタンス側からイベントをインタフェースオブジェクトに伝える過程を示すフローチャートである。

【図５】本発明の制御システムによって構築されたＰＯＳシステムの周辺装置を制御するシステムを示すブロック図である。

【符号の説明】

１０・・第１のＯＣＸ

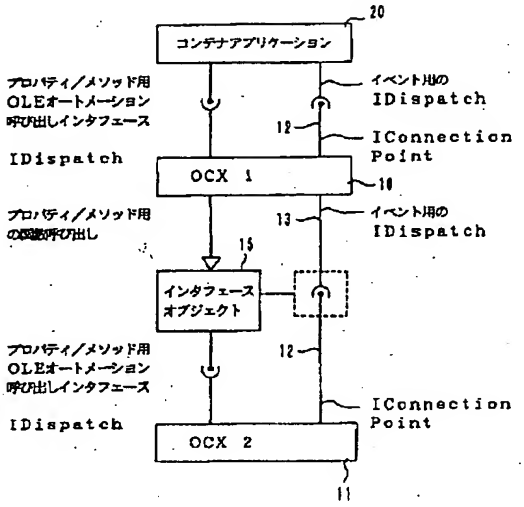
21

１１・・第２のＯＣＸ
１５・・インタフェースオブジェクト
２０・・アプリケーション
７０・・パソコン
７１・・ＰＯＳアプリケーションプログラム
７２・・制御オブジェクトによって構成された制御システム

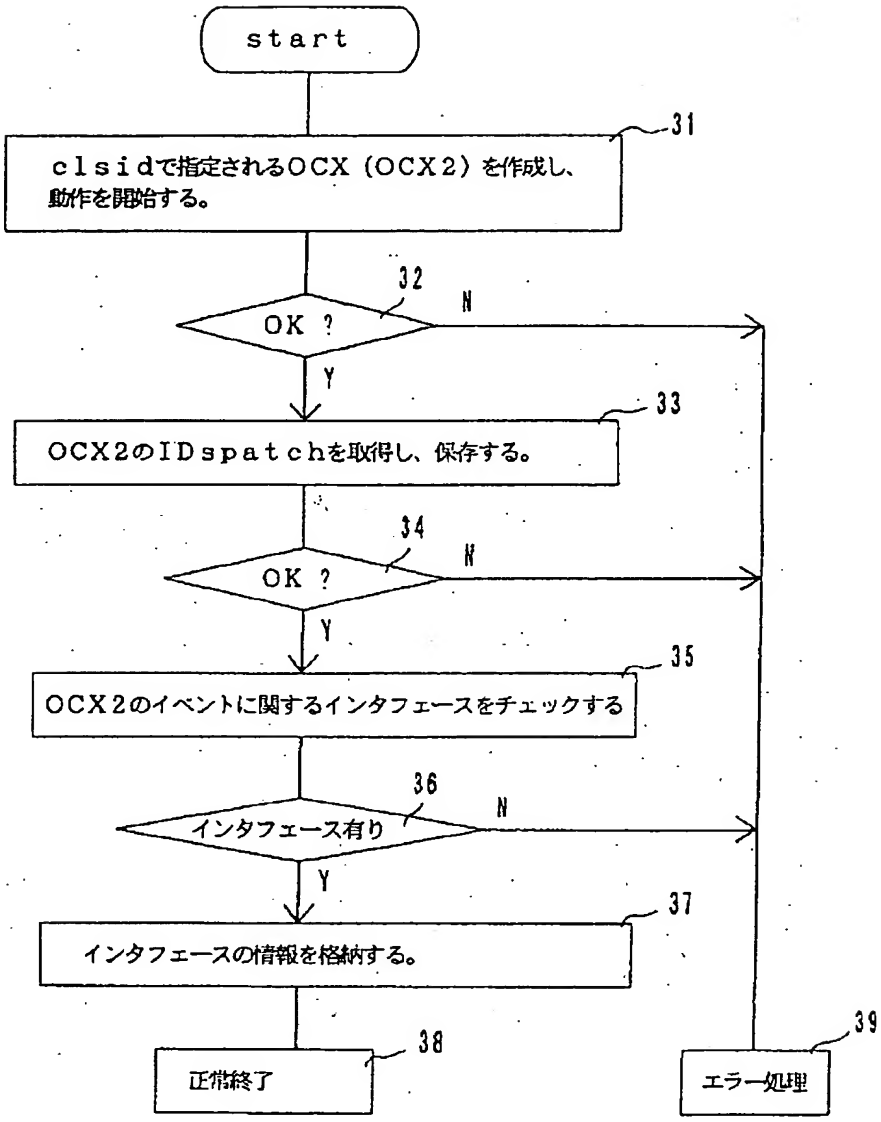
22

１００・・ポートドライバ
１１０・・カスタマディスプレイ
１１１・・プリンタ
１１２・・レシート印字機構
１１３・・スリップ印字機構
１１５・・キャッシュドロワ

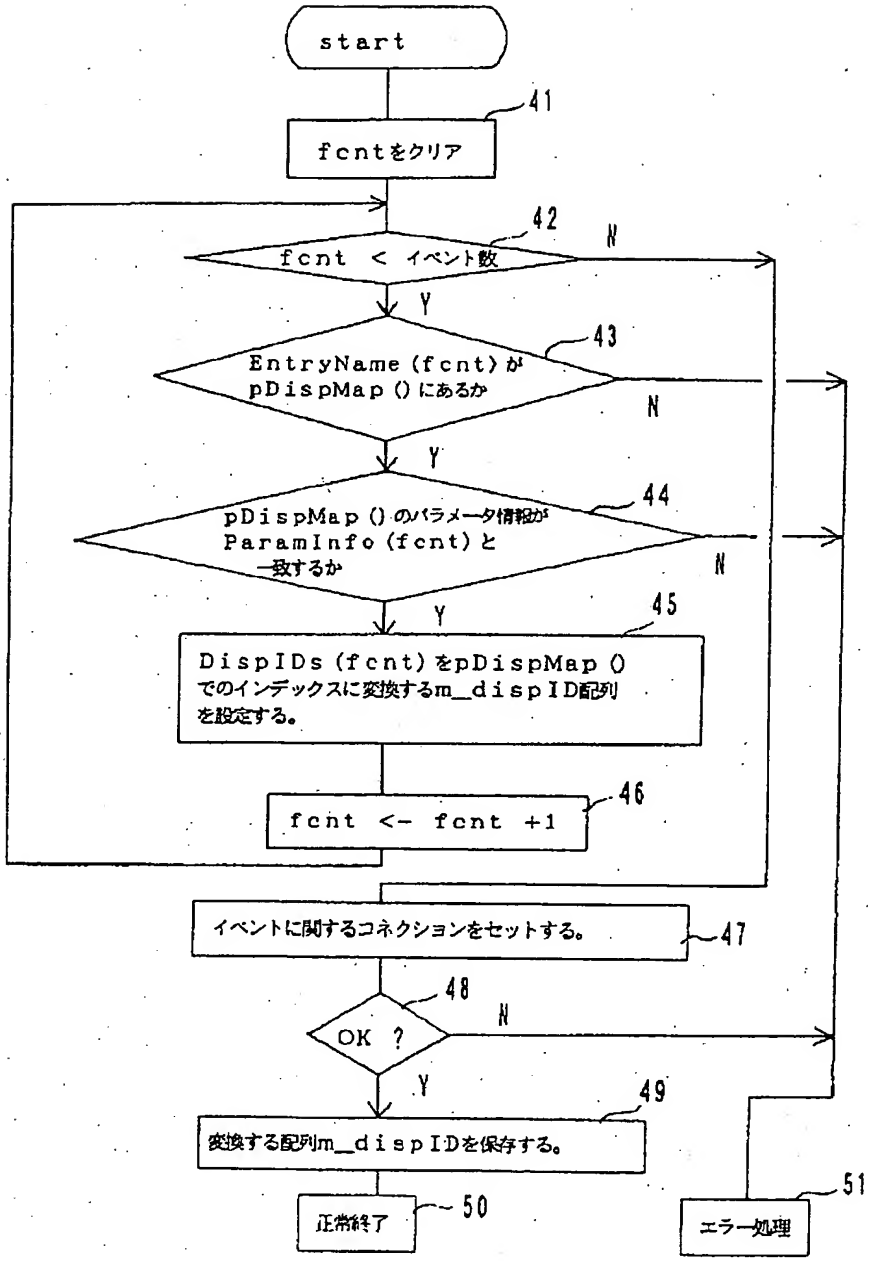
【図１】



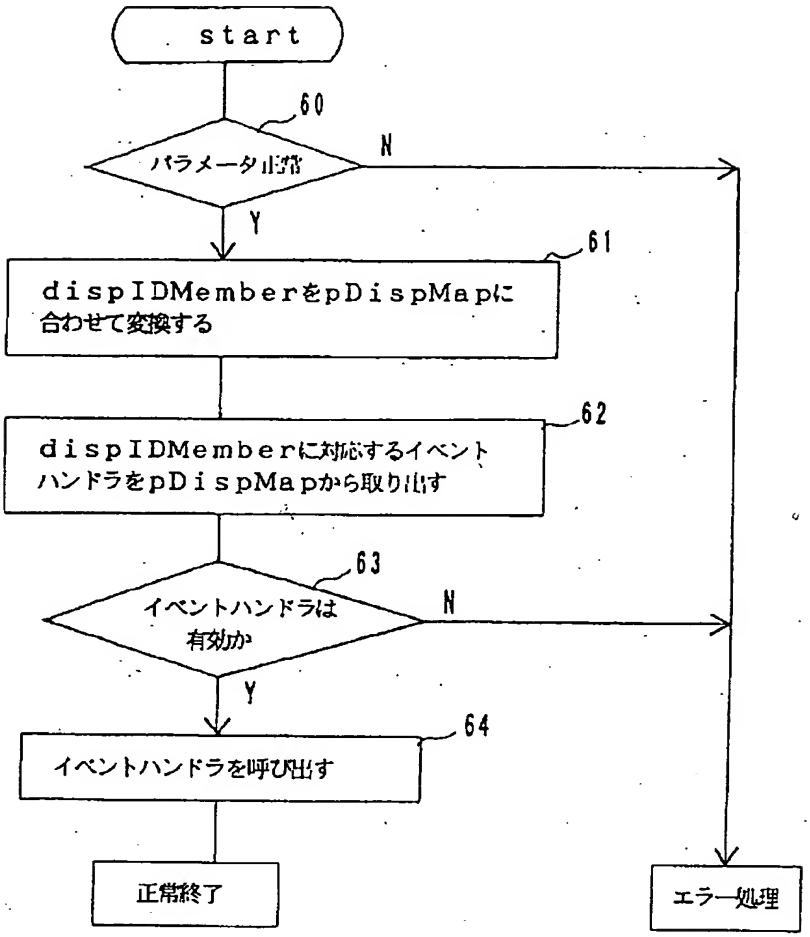
【図２】


start
31
ｃｌｓｉｄで指定されるＯＣＸ（ＯＣＸ２）を作成し、動作を開始する。
32
OK ?
N
Y
33
ＯＣＸ２のＩＤｓｐａｔｃｈを取得し、保存する。
34
OK ?
N
Y
35
ＯＣＸ２のイベントに関するインタフェースをチェックする
36
インタフェース有り
N
Y
37
インタフェースの情報を格納する。
38
正常終了
39
エラー処理

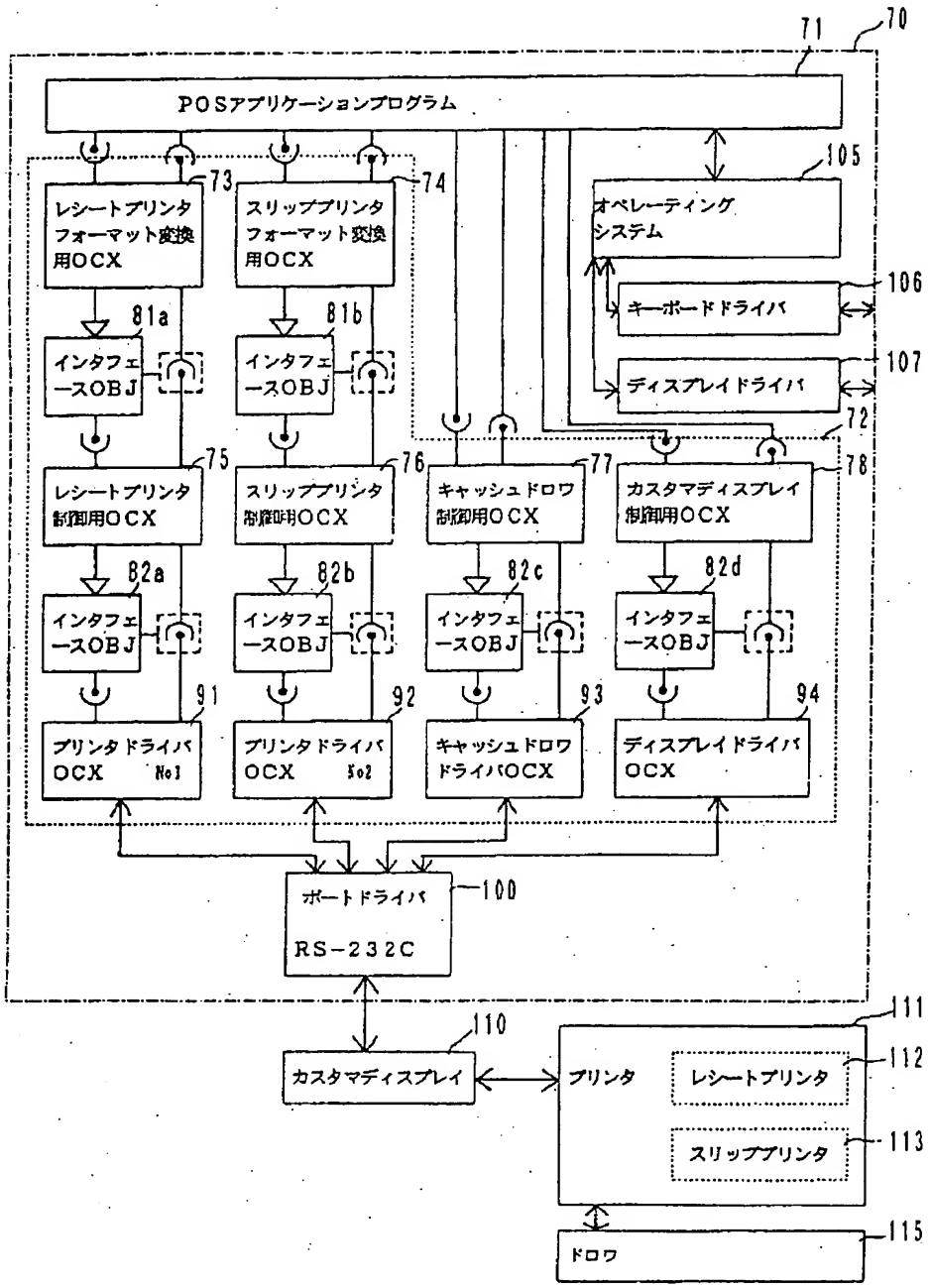
【図３】


start
41
fcntをクリア
42
fcnt ＜ イベント数
N
Y
43
EntryName (fcnt) が
pDispMap () にあるか
N
Y
44
pDispMap () のパラメータ情報が
ParamInfo (fcnt) と
一致するか
N
Y
45
DispIDs (fcnt) をpDispMap ()
でのインデックスに変換するm_dispID配列
を設定する。
46
fcnt <- fcnt +1
イベントに関するコネクションをセットする。
47
48
OK ?
N
Y
49
変換する配列m_dispIDを保存する。
50
正常終了
51
エラー処理

【図４】


start
60
パラメータ正常
N
Y
61
dispIDMemberをpDispMapに
合わせて変換する
62
dispIDMemberに対応するイベント
ハンドラをpDispMapから取り出す
63
イベントハンドラは
有効か
N
Y
64
イベントハンドラを呼び出す
正常終了
エラー処理

【図５】


71
70
POSアプリケーションプログラム
105
オペレーティング
システム
106
キーボードドライバ
107
ディスプレイドライバ
72
73
レシートプリンタ
フォーマット変換
用OCX
74
スリッププリンタ
フォーマット変換
用OCX
81a
インタフェ
ースOBJ
81b
インタフェ
ースOBJ
75
レシートプリンタ
制御用OCX
76
スリッププリンタ
制御用OCX
77
キャッシュドロワ
制御用OCX
78
カスタマディスプレイ
制御用OCX
82a
インタフェ
ースOBJ
82b
インタフェ
ースOBJ
82c
インタフェ
ースOBJ
82d
インタフェ
ースOBJ
91
プリンタドライバ
OCX No1
92
プリンタドライバ
OCX No2
93
キャッシュドロワ
ドライバOCX
94
ディスプレイドライバ
OCX
100
ポートドライバ
RS-232C
110
カスタマディスプレイ
111
プリンタ
112
レシートプリンタ
113
スリッププリンタ
115
ドロワ